CHAPTER

Domain 6: Security assessment and testing

6

CHAPTER OUTLINE

INTRODUCTION

Security assessment and testing are critical components of any information security program. Organizations must accurately assess their real-world security, focus on the most critical components, and make necessary changes to improve.

In this domain, we will discuss two major components of assessment and testing: overall security assessments, including vulnerability scanning, penetration testing, and security audits; and testing software via static and dynamic methods.

Eleventh Hour CISSP®. http://dx.doi.org/10.1016/B978-0-12-811248-9.00006-1

ASSESSING ACCESS CONTROL

A number of processes exist to assess the effectiveness of access control. Tests with a narrower scope include penetration tests, vulnerability assessments, and security audits. A security assessment is a broader test that may include narrower tests, such as penetration tests, as subsections.

PENETRATION TESTING

A penetration tester is a white hat hacker who receives authorization to attempt to break into an organization's physical or electronic perimeter (sometimes both). *Penetration tests* (called "pen tests" for short) are designed to determine whether black hat hackers could do the same. They are a narrow but often useful test, especially if the penetration tester is successful.

Penetration tests may include the following tests:

- Network (Internet)
- Network (internal or DMZ)
- War dialing
- Wireless
- Physical (attempt to gain entrance into a facility or room)

Network attacks may leverage client-side attacks, server-side attacks, or Web application attacks. *War dialing*, which gets its name from the 1983 movie *WarGames*, uses a modem to dial a series of phone numbers, looking for an answering modem carrier tone. The penetration tester then attempts to access the answering system.

Social engineering is a no-tech or low-tech method that uses the human mind to bypass security controls. Social engineering may be used in combination with many types of attacks, especially client-side attacks or physical tests. An example of a social engineering attack combined with a client-side attack is emailing malware with a subject line of "Category 5 Hurricane is about to hit Florida!"

A *zero-knowledge* test, also called black-box test, is "blind"; the penetration tester begins with no external or trusted information and begins the attack with public information only. A *full-knowledge test* (also called *crystal-box test*) provides internal information to the penetration tester, including network diagrams, policies and procedures, and sometimes reports from previous penetration testers. *Partial-knowledge* tests are in between zero and full knowledge; the penetration tester receives some limited trusted information.

Penetration testing tools and methodology

Penetration testers often use penetration testing tools, which include the open-source Metasploit (http://www.metasploit.org), and closed-source Core Impact (http://www.coresecurity.com), and Immunity Canvas (http://www.immunitysec.com). Pen testers also use custom tools, as well as malware samples and code posted to the Internet.

Penetration testers use the following methodology:

- Planning
- Reconnaissance
- Scanning (also called enumeration)
- Vulnerability assessment
- Exploitation
- Reporting

Black hat hackers typically follow a similar methodology although they may perform less planning and obviously omit reporting. Black hats will also cover their tracks by erasing logs and other signs of intrusion, and they frequently violate system integrity by installing back doors in order to maintain access. A penetration tester should always protect data and system integrity.

Assuring confidentiality, data integrity, and system integrity

Penetration testers must ensure the confidentiality of any sensitive data that is accessed during the test. If the target of a penetration test is a credit card database, the penetration tester may have no legal right to view or download the credit cards. Testers will often request that a dummy file containing no regulated or sensitive data be placed in the same area of the system as the credit card data and protected with the same permissions. If the tester can read and/or write to that file, then they prove they could have done the same to the credit card data.

Penetration testers must ensure the system integrity and data integrity of their client's systems. Any active attack, as opposed to a passive read-only attack, against a system could potentially cause damage; this can be true even for an experienced penetration tester. This risk must be clearly understood by all parties, and tests are often performed during change maintenance windows for this reason.

One potential issue that should be discussed before the penetration test commences is the risk of encountering signs of a previous or current successful malicious attack. Penetration testers sometimes discover that they are not the first attacker to compromise a system and that someone has beaten them to it. Attackers will often become more malicious if they believe they have been discovered, sometimes violating data and system integrity. The integrity of the system is at risk in this case, and the penetration tester should end the penetration test and immediately escalate the issue.

Finally, the final penetration test report should be protected at a very high level, as it contains a roadmap to attack the organization.

VULNERABILITY TESTING

Vulnerability scanning or vulnerability-testing scans a network or system for a list of predefined vulnerabilities such as system misconfiguration, outdated software, or a lack of patching. A vulnerability-testing tool such as Nessus (http://www.tenable.com/products/nessus-vulnerability-scanner) or OpenVAS (http://www.openvas.org) may be used to identify the vulnerabilities.

SECURITY AUDITS

A *security audit* is a test against a published standard. Organizations may be audited for Payment Card Industry Data Security Standard (PCI DSS) compliance (discussed in Chapter 3), for example. PCI DSS includes many required controls, such as firewalls, specific access control models, and wireless encryption. An auditor then verifies that a site or organization meets the published standard.

SECURITY ASSESSMENTS

Security assessments are a holistic approach to assessing the effectiveness of access control. Instead of looking narrowly at penetration tests or vulnerability assessments, security assessments have a broader scope.

Security assessments view many controls across multiple domains and may include the following:

- Policies, procedures, and other administrative controls
- Assessing the real world-effectiveness of administrative controls
- Change management
- Architectural review
- Penetration tests
- Vulnerability assessments
- Security audits

As the above list shows, a security assessment may include other distinct tests, such as a penetration test. The goal is to broadly cover many other specific tests to ensure that all aspects of access control are considered.

LOG REVIEWS

Reviewing security audit logs within an IT system is one of the easiest ways to verify that access control mechanisms are performing adequately. Reviewing audit logs is primarily a detective control.

The intelligence gained from proactive audit log management and monitoring can be very beneficial; the collected antivirus logs of thousands of systems can give a very accurate picture of the current state of malware. Antivirus alerts combined with a spike in failed authentication alerts from authentication servers or a spike in outbound firewall denials may indicate that a password-guessing worm is attempting to spread across a network.

SOFTWARE TESTING METHODS

There is a variety of software testing methods. In addition to testing the features and stability of the software, testing increasingly focuses on discovering specific

programmer errors leading to vulnerabilities that risk system compromise, including a lack of bounds checking. Two general approaches to automated code review exist: static and dynamic testing.

STATIC AND DYNAMIC TESTING

Static testing tests the code passively; the code is not running. This includes walk-throughs, syntax checking, and code reviews. Static analysis tools review the raw source code itself looking for evidence of known insecure practices, functions, libraries, or other characteristics used in the source code. The Unix lint program performs static testing for C programs.

Dynamic testing tests the code while executing it. With dynamic testing, security checks are performed while actually running or executing the code or application under review.

Both approaches are appropriate and complement each other. Static analysis tools might uncover flaws in code that have not even yet been fully implemented in a way that would expose the flaw to dynamic testing. However, dynamic analysis might uncover flaws that exist in the particular implementation and interaction of code that static analysis missed.

White-box software testing gives the tester access to program source code, data structures, variables, etc. *Black-box testing* gives the tester no internal details; the software is treated as a black box that receives inputs.

TRACEABILITY MATRIX

A *traceability matrix*, sometimes called a requirements traceability matrix (RTM), can be used to map customers' requirements to the software testing plan; it traces the requirements and ensures that they are being met. It does this by mapping customer usage to test cases. Fig. 6.1 shows a sample RTM.

Requirement ID	Requirements tested	Use case 1.1	Use case 1.2	Use case 1.3	Use case 1.4	…
Test cases	34	2	4	7	5	
TC1.1.1	2	X		X		
TC1.1.2	1				X	
TC1.2.1	3	X		X	X	
TC1.2.2	1		X			
TC1.2.3	2			X	X	
…						

FIG. 6.1

Sample requirements traceability matrix.[1]

SYNTHETIC TRANSACTIONS

Synthetic transactions, or synthetic monitoring, involve building scripts or tools that simulate activities normally performed in an application. The typical goal of using synthetic transactions/monitoring is to establish expected norms for the performance of these transactions. These synthetic transactions can be automated to run on a periodic basis to ensure the application is still performing as expected. These types of transactions can also be useful for testing application updates prior to deployment to ensure that functionality and performance will not be negatively impacted. This type of testing or monitoring is most commonly associated with custom-developed web applications.

SOFTWARE TESTING LEVELS

It is usually helpful to approach the challenge of testing software from multiple angles, addressing various testing levels from low to high. The software testing levels designed to accomplish that goal are unit testing, installation testing, integration testing, regression testing, and acceptance testing.

FAST FACTS

- *Unit testing*: Low-level tests of software components, such as functions, procedures, or objects.
- *Installation testing*: Testing software as it is installed and first operated.
- *Integration testing*: Testing multiple software components as they are combined into a working system. Subsets may be tested, or *Big Bang* integration testing is used for all integrated software components.
- *Regression testing*: Testing software after updates, modifications, or patches.
- *Acceptance testing*: Testing to ensure that the software meets the customer's operational requirements. When this testing is done directly by the customer, it is called user acceptance testing.

FUZZING

Fuzzing (also called *fuzz testing*) is a type of black-box testing that submits random, malformed data as inputs into software programs to determine if they will crash. A program that crashes when receiving malformed or unexpected input is likely to suffer from a boundary-checking issue and may be vulnerable to a buffer overflow attack.

Fuzzing is typically automated, repeatedly presenting random input strings as command line switches, environment variables, and program inputs. Any program that crashes or hangs has failed the fuzz test.

COMBINATORIAL SOFTWARE TESTING

Combinatorial software testing is a black-box testing method that seeks to identify and test all unique combinations of software inputs. An example of combinatorial software testing is *pairwise testing*, also called *all-pairs testing*.

MISUSE CASE TESTING

Misuse case testing leverages use cases for applications, which spell out how various functionalities will be leveraged within an application. Formal use cases are typically built as a flow diagram written in UML (Unified Modeling Language) and are created to help model expected behavior and functionality.

Misuse case testing models how a security impact could be realized by an adversary abusing the application. This can be seen simply as a different type of use case, but the reason for calling out misuse case testing specifically is to highlight the general lack of attacks against the application.

TEST COVERAGE ANALYSIS

Test or code coverage analysis attempts to identify the degree to which code testing applies to the entire application. The goal is to ensure that there are no significant gaps where a lack of testing could allow for bugs or security issues to be present that otherwise should have been discovered.

INTERFACE TESTING

Interface testing is primarily concerned with appropriate functionality being exposed across all the ways users can interact with the application. From a security-oriented vantage point, the goal is to ensure that security is uniformly applied across the various interfaces. This type of testing exercises the various attack vectors an adversary could leverage.

SUMMARY OF EXAM OBJECTIVES

In this domain, we have learned about various methods to test real-world security of an organization, including vulnerability scanning, penetration testing, security assessments, and audits. Vulnerability scanning determines one half of the Risk = Threat × Vulnerability equation. Penetration tests seek to match those vulnerabilities with threats in order to demonstrate real-world risk. Assessments provide a broader view of the security picture, and audits demonstrate compliance with a published specification, such as PCI DSS.

We discussed testing code security, including static methods such as source code analysis, walkthroughs, and syntax checking. We discussed dynamic methods used on running code, including fuzzing and various forms of black-box testing. We also discussed synthetic transactions, which attempt to emulate real-world use of an application through the use of scripts or tools that simulate activities normally performed in an application.

TOP FIVE TOUGHEST QUESTIONS

1. What can be used to ensure that software meets the customer's operational requirements?
   a. Integration testing
   b. Installation testing
   c. Acceptance testing
   d. Unit testing
2. What term describes a black-box testing method that seeks to identify and test all unique combinations of software inputs?
   a. Combinatorial software testing
   b. Dynamic testing
   c. Misuse case testing
   d. Static testing

   Use the following scenario to answer questions 3–5:

   You are the CISO (chief information security officer) of a large bank and have hired a company to provide an overall security assessment, as well as complete a penetration test of your organization. Your goal is to determine overall information security effectiveness. You are specifically interested in determining if theft of financial data is possible.

   Your bank has recently deployed a custom-developed, three-tier web application that allows customers to check balances, make transfers, and deposit checks by taking a photo with their smartphone and then uploading the check image. In addition to a traditional browser interface, your company has developed a smartphone app for both Apple iOS and Android devices.

   The contract has been signed, and both scope and rules of engagement have been agreed upon. A 24/7 operational IT contact at the bank has been made available in case of any unexpected developments during the penetration test, including potential accidental disruption of services.
3. Assuming the penetration test is successful, what is the best way for the penetration testing firm to demonstrate the risk of theft of financial data?
   a. Instruct the penetration testing team to conduct a thorough vulnerability assessment of the server containing financial data.
   b. Instruct the penetration testing team to download financial data, redact it, and report accordingly.
   c. Instruct the penetration testing team that they may only download financial data via an encrypted and authenticated channel.
   d. Place a harmless “flag” file in the same location as the financial data, and inform the penetration testing team to download the flag.
4. You would like to have the security firm test the new web application, but have decided not to share the underlying source code. What type of test could be used to help determine the security of the custom web application?
   a. Secure compiler warnings

   b. Fuzzing
   c. Static testing
   d. White-box testing
5. During the course of the penetration test, the testers discover signs of an active compromise of the new custom-developed, three-tier web application. What is the best course of action?
   a. Attempt to contain and eradicate the malicious activity.
   b. Continue the test.
   c. Quietly end the test, immediately call the operational IT contact, and escalate the issue.
   d. Shut the server down.

ANSWERS

1. Correct answer and explanation: C. Acceptance testing is designed to ensure the software meets the customer's operational requirements.
   Incorrect answers and explanations: Answers A, B, and D are incorrect. Integration testing examines multiple software components as they are combined into a working system. Installation testing examines software as it is installed and first operated. Unit testing is a low-level test of software components, such as functions, procedures, or objects.
2. Correct answer and explanation: A. Combinatorial software testing is a black-box testing method that seeks to identify and test all unique combinations of software inputs.
   Incorrect answers and explanations: Answers B, C, and D are incorrect. Dynamic testing examines code while executing it. Misuse case testing formally models how security would be impacted by an adversary abusing the application. Static testing examines the code passively; the code is not running. This form of testing includes walkthroughs, syntax checking, and code reviews.
3. Correct answer and explanation: D. A flag is a dummy file containing no regulated or sensitive data. It is placed in the same area of the system as the credit card data and protected with the same permissions. If the tester can read and/or write to that file, then they prove they could have done the same to the credit card data.
   Incorrect answers and explanations: Answers A, B, and C are incorrect. Answer A is a vulnerability assessment, not a penetration test. Answers B and C are dangerous and could involve unauthorized access of regulated data, such as health care records.
4. Correct answer and explanation: B. Fuzzing is a black-box testing method that does not require access to source code.
   Incorrect answers and explanations: Answers A, C, and D are incorrect. All are static methods that require access to source code.

5. Correct answer and explanation: C. Attackers will often act more maliciously if they believe they have been discovered, sometimes violating data and system integrity. The integrity of the system is at risk in this case, and the penetration tester should end the penetration test and immediately escalate the issue. Incorrect answers and explanations: Answers A, B, and D are incorrect. The client must be notified immediately, as incident handling is not the penetration tester's responsibility.

ENDNOTE

1. *Combinatorial software testing*. http://csrc.nist.gov/groups/SNS/acts/documents/kuhn-kacker-lei-hunter09.pdf [accessed 25.04.16].